Getting Started in R

Part 3

by M. Papathomas and E. Rexstad (with material from a document prepared by R. King)

```
## Warning: package 'knitr' was built under R version 3.2.5
```

Solutions to questions

In the previous Microlab session, we wrote functions in `R`. In this practical, we will look at understanding a given piece of `R` code used to solve a specific problem problem. This will be extended to modify the code to undertake changes in the problem being considered.

Example: Playing a game

You are offered the chance to take part in a game with the following rules:

- You pay £5 to play.
- You throw five dice.
- Any that do not show a six, you throw again.
- Any that still do not show a six, you throw a third time.
- Your winnings are £2^n where n is the final number of sixes. (Your profit is therefore £$(2^n - 5)$.)

Under these rules, determine the following consequences of playing this dice game.

- You play the game once. What is the probability that you throw no sixes at the first throw of five dice? Hence or otherwise, find an expression for the probability that you throw no sixes at all. Use `R` to evaluate your answer to four decimal places.

Solution

Let X be the number of sixes thrown at the first throw of the five dice. Then, the probability of throwing no sixes after the first throw is given by

$$\mathbb{P}(X = 0) = \left(\frac{5}{6}\right)^5 = 0.4019,$$

because the probability of not throwing a 6 with any particular die is $\frac{5}{6}$ and the throws are independent of each other. The probability of throwing no sixes at all is simply the probability we throw no sixes in all three turns of the game (which all involve independent throws of the dice), so that this is equal to $(\mathbb{P}(X = 0))^3 = 0.0649$.

In R,

```r
((5/6)^5)^3
```

```
## [1] 0.06490547
```

- Explain carefully what the following `R` code does, relating the `R` commands to the rules of the game.

```r
psixes <- rep(0,6)
for (k in 1:1000){
  sixes <- rep(0,5)
  for (i in 1:3){
    for (j in 1:5){
      if(sixes[j]==0){
          x <- rbinom(1,1,1/6)
          if(x==1) sixes[j] <- 1
      }
    }
  }
  y <- sum(sixes)
  psixes[y+1] <- psixes[y+1]+1
}
psixes
```

Solution

In addition to providing a description of the code, I have also converted it into a function because we will be using the function repeatedly.

```r
dice.game <- function(numgames=1000, numdice=5, numturns=3) {
  psixes <- rep(0,6)  # This sets up psixes as a vector of length 6 with
                      # each element equal to 0; it is used to count the
                      # number of sixes we obtain when repeatedly playing the game.
                      # Note that the yth element corresponds to (y-1) sixes
                      # thrown (there is no 0 element to a vector)
                      # e.g. when psixes is (0,0,15,0,0,0) we got 2 sixes 15 times

  for (k in 1:numgames){  # A loop to play the game a total of numgames times.

  sixes <- rep(0,numdice)   # This sets up sixes as a vector of length numdice
                            # with each element equal to 0;
                            # It is used to identify whether a six has been thrown
                            # by each individual die.

  for (i in 1:numturns){     # Loop over the numturns turns in the game
      for (j in 1:numdice){  # Loop over each single die thrown in the game
          if(sixes[j]==0){   # This is used to check if the number of the die is a 6
                             # If equal to 0 no 6 on die and we throw the die again;
                             # If equal to 1 a 6 has been thrown on the die.

              x <- rbinom(1,1,1/6)    # We thrown die j once and record whether we
                                      # thrown a 6 (a success);
                                      # We throw a 6 with probability 1/6 and record
                                      # x = 1; else x = 0.
              if(x==1) sixes[j] <- 1  # If we throw a 6, we update the jth element of
                                      # sixes to record that a 6 has been thrown by die j.
          }                     # End if (if we throw the die - dependent if it shows a six).
      }                         # End loop when all die have been ``thrown'' (end of a turn)
  }                             # End loop when all turns have passed (end of a game)
  y <- sum(sixes)               # Add up the elements of the vector sixes corresponding
                                # to the number of sixes thrown
  psixes[y+1] <- psixes[y+1]+1    # Update the number times that we have observed y sixes
```

```
                                    # The (y+1)th element of psixes corresponds to y sixes.

  }                                 # End loop when played game numgames times.
  return(psixes)                    # Output total number of sixes thrown for numgames games played
}
```

- Type the preceeding code into the R-Studio editor panel, submit it to the R console and use it to check your manual answer from the previous part. HINT: Until you are confident of what your code is doing, it may be better to substitute `for (k in 1:100){` for the line `for (k in 1:1000){` (resulting in precision to only 3 decimal places). If you are unsure of what an `R` command does, use the `help()` function.

Solution

Running my function produces

```
result <- dice.game()
result
```

```
## [1]  63 226 353 245 102  11
```

Out of 1000 games, I observed 63 games with no sixes. The estimate of the probability of throwing no sixes would be 63/1000 = 0.063, reasonably close to the answer derived in the previous section.

- Given your output obtained from running the code, provide an estimate of the probability mass function for
  - number of sixes thrown and
  - profit.

Solution

Dividing the output of my function by `numgames` (i.e. 1000), produces

x	1.000	2.000	3.000	4.000	5.000	6.000
fx(x)	0.063	0.226	0.353	0.245	0.102	0.011

y	-4.000	-3.000	-1.000	3.000	11.000	27.000
fy(y)	0.063	0.226	0.353	0.245	0.102	0.011

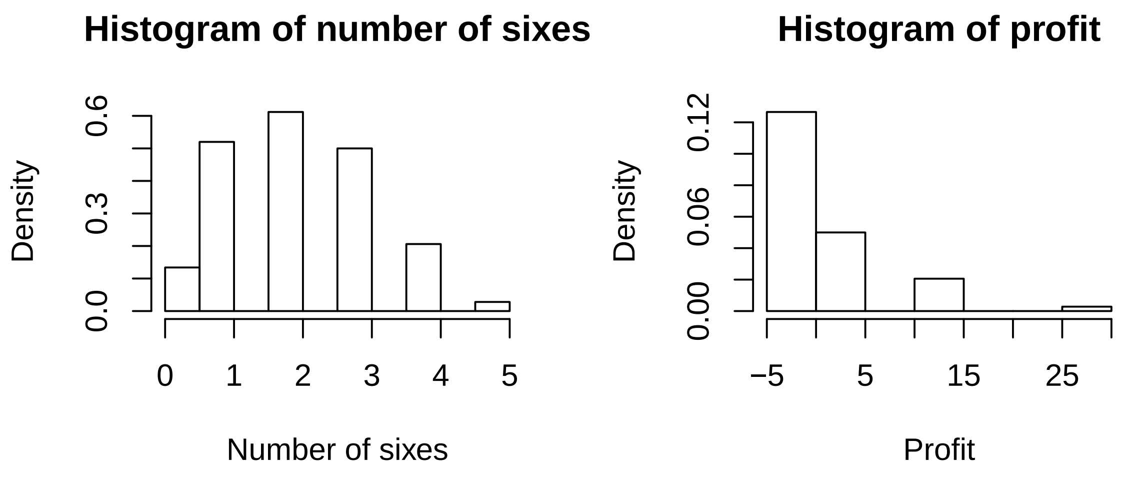
In the second table, winnings exceed the original stake when profit exceed 0. Summing the probabilities of the final three entries produces 0.245 + 0.102 + 0.011 = 0.358

- Calculate an estimate of the probability that your winnings exceed the original £5 if the game is played a single time.
- Extend your `R` code to plot histograms of the estimated probability mass function of
  - number of sixes thrown and
  - profit.
  - HINT: the variable `psixes` is a set of six frequencies that sum to 1000. To plot a histogram of the number of sixes over 1000 games, you need to set up a variable with 1000 observations; each of these observations will be one of the numbers 0,1,2,3,4,5. The easy way to do this is to store the 1000 values in a vector. For your histograms, you may find the options `freq` and `breaks` useful — see `help(hist)`.

Solution

Revised function:

```
dice.game.thrown <- function(numgames=1000, numdice=5, numturns=3) {
  psixes <- rep(0,6)
  nsixes <- rep(0, numgames)  # new for holding number of sixes in each game
  for (k in 1:numgames){
    sixes <- rep(0,numdice)
    for (i in 1:numturns){
      for (j in 1:numdice){
        if(sixes[j]==0){
            x <- rbinom(1,1,1/6)
            if(x==1) sixes[j] <- 1
        }
      }
    }
    y <- sum(sixes)
    psixes[y+1] <- psixes[y+1]+1
    nsixes[k] <- y          # new line for number sixes in each game
  }
  par(mfrow=c(1,2))
  hist(nsixes,xlab="Number of sixes",probability="TRUE",
       main="Histogram of number of sixes")      # New line added
  hist(2^nsixes-5,xlab="Profit",probability="TRUE",
       main="Histogram of profit")
  return(psixes)
}
more.results <- dice.game.thrown()
```



Only the x-axis changed between the histograms because *profit* is simply a transformation of the number of sixes thrown. The histograms can be tuned to have more bars depicted by adding the argument `nbreaks=` to the `hist()` function call.

- Further extend your `R` code to obtain an empirical estimate of the expected number of sixes thrown and the expected profit, by using the estimated probability mass function and the definition of expectation.

Solution

Using the results of our computation, stored in the object `more.results` along with the definition of expection, we can compute $\mathbb{E}(X) = \sum_x x f_X(x)$ with this code

```
values <- seq(from=0, to=5)     # do not use seq(0:5), causes subtle error
expected.sixes <- sum(values*more.results/1000)
expected.profit <- sum((2^values-5)*more.results/1000)
print(paste("Expected value of number of sixes = ", expected.sixes))
```

```
## [1] "Expected value of number of sixes =  2.104"
```

```
print(paste("Expected value of profit = ", expected.profit))
```

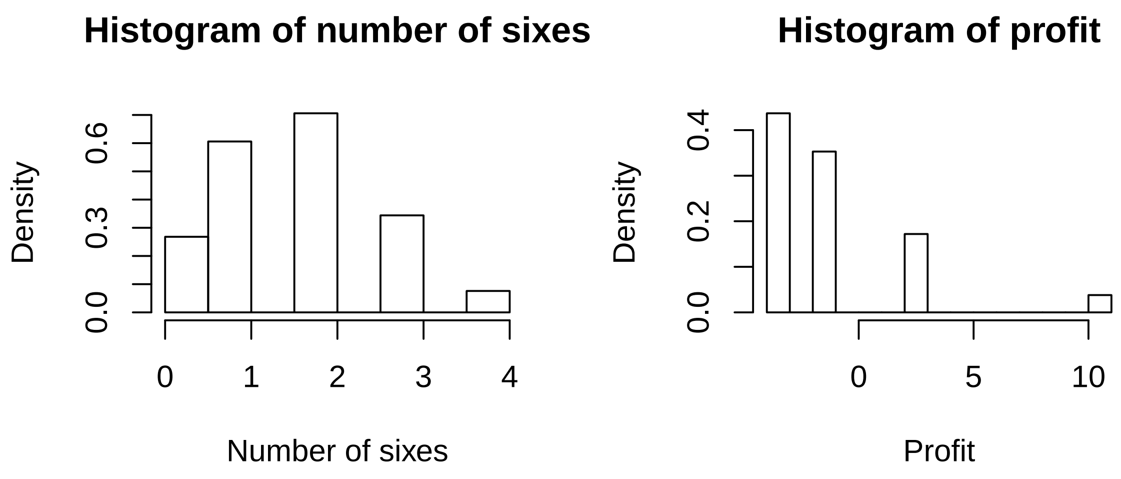
```
## [1] "Expected value of profit =  0.907"
```

- Modify the code to play the game with only 4 dice. How does this affect the expected number of sixes thrown, expected profit and probability of your winnings exceeding the original £5 stake?

Solution

This should be wonderfully easy to do, given that we have modified our code to perform as a function. To explore the 4-dice game, we merely alter the arguments in our function.

```r
four.dice.results <- dice.game.thrown(numdice=4)  # other two arguments remain at default values
```



```r
print(four.dice.results)
```

```
## [1] 134 303 353 172  38   0
```

```r
four.expected.sixes <- sum(values*four.dice.results/1000)
four.expected.profit <- sum((2^values-5)*four.dice.results/1000)
prob.profit <- (four.dice.results[4] + four.dice.results[5])/1000
print(paste("E(sixes)=", four.expected.sixes, "E(profit)=", four.expected.profit,
            "Pr(profit)=", prob.profit))
```

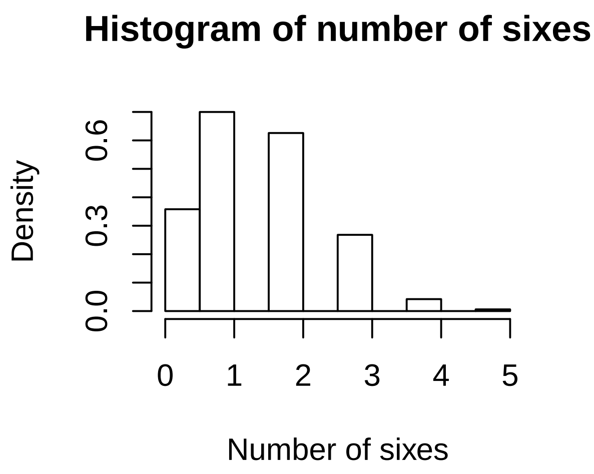
```
## [1] "E(sixes)= 1.677 E(profit)= -0.864 Pr(profit)= 0.21"
```

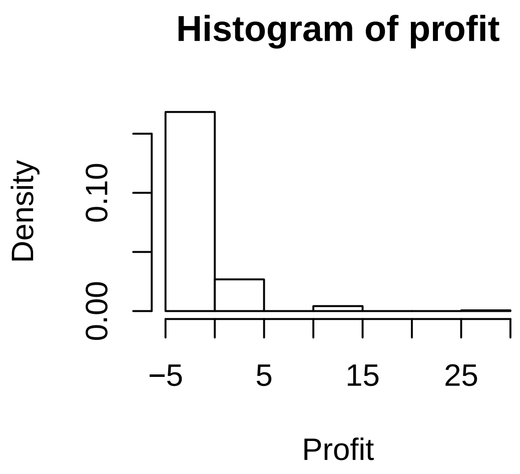
- Reverting back to the five dice, explore the game where you are only throw the dice for a total of two times. How does this affect the expected number of sixes thrown, expected profit and probability of your winnings exceeding the original £5 stake?

Solution

Simple to find this solution by changing the arguments of our function

```r
two.turn.results <- dice.game.thrown(numturns = 2)  # other two arguments remain at default values
```





```r
print(two.turn.results)
```

```
## [1] 179 350 313 134  21   3
```

```r
two.turn.expected.sixes <- sum(values*two.turn.results/1000)
two.turn.expected.profit <- sum((2^values-5)*two.turn.results/1000)
prob.profit <- (two.turn.results[4] + two.turn.results[5] + two.turn.results[6])/1000
print(paste("E(sixes)=", two.turn.expected.sixes, "E(profit)=", two.turn.expected.profit,
            "Pr(profit)=", prob.profit))
```

```
## [1] "E(sixes)= 1.477 E(profit)= -1.365 Pr(profit)= 0.158"
```

[You can see our function could stand some further revisions to calculate the set of expected values, but that is left for you.]

Generalisation we can make regarding the variants of this dice game.

- Only the original game (3 turns with 5 dice) had a positive expected profit.
- This version, with the positive expectation, had a single game success probability of roughly $\frac{1}{3}$, meaning a large number of games would be needed to reach the positive expected profit.
- By either reducing the number of dice or the number of turns, the expected profit becomes negative and playing the game would not be good for the bank account.
  - Profit is more greatly reduced by reducing the number of turns in comparison to reducing the number of dice.
